# End to End Encryption Chat

CECS 478
Game of Threads
Raymond Chin & Michael Ly

# Content

-Introduction

-Implementation

-Demo

-Attack Surfaces

-Future Work

# Encryptor

```
key = os.urandom(32)
IV = 16 * '\x00'
mode = AES.MODE_CBC
encryptor = AES.new(key, mode, IV=IV)

#need padding because IV only works for code in multiples of 16
 paddingAES = random.choice(letters)
 extra = len(plaintext) % 16
 if extra > 0:
     plaintext = plaintext + (paddingAES * (16 - extra))
padder = padding.PKCS7(128).padder()
plaintext = plaintext.encode('utf-8')
plaintext = padder.update(plaintext)+padder.finalize()
##encrpyt plaintext with AES
ciphertext = encryptor.encrypt(plaintext)
#test to make sure AES works
#return (ciphertext, IV)
#creating HMAC key
secret = os.urandom(32)
h = hmac.HMAC(secret, hashes.SHA256(), backend=default_backend())
h.update(ciphertext)
#get tag to return
tag = h.finalize()
#holds aes and hmac key
combinedKey = key + secret
#Encrypt using concatenated keys using RSA
cipher_rsa = PKCS1_OAEP.new(publicKey)
RSACipher = cipher_rsa.encrypt(combinedKey)

return(b64encode(RSACipher).decode('utf-8'),b64encode(ciphertext).decode('utf-8'), b64encode(tag).decode('utf-8'),IV)
```

# Decryptor

```
RSACipher = b64decode(RSAC)
ciphertext = b64decode(ct)
tag = b64decode(tg)
#IV = getJSON('./Encryption.json').get('IV',[])
#Decrypt RSA combined'
cipher_rsa = PKCS1_OAEP.new(pathtoPrivate)
combinedKey = cipher_rsa.decrypt(RSACipher)

#The first 32 are the key for AES last is the HMAC
key = combinedKey[0:32]
HMACkey = combinedKey[32:64]

#First need to compare tag to ensure authenticity
h = hmac.HMAC(HMACkey, hashes.SHA256(), backend = default_backend())
h.update(ciphertext)
#test tag to prove it works ;)
badTag =("well this is blatantly wrong")
badTag = bytes(badTag, 'utf-8')
h.verify(tag)
#If it passes it won't say anything however if it doesn't
#it won't continue with the code instead it stops and
#says "Signature did not match digest"
#print("Tag verified! Signature matches! No shenanigans going on here.")

#Decrypt AES
mode = AES.MODE_CBC
decryptor = Cipher(algorithms.AES(key),modes.CBC(IV.encode()),default_backend()).decryptor()
plaintext = decryptor.update(ciphertext) + decryptor.finalize()
#Unpadding plaintext
unpadder = padding.PKCS7(128).unpadder()
plaintext = unpadder.update(plaintext) + unpadder.finalize()

finaltext = plaintext.decode("utf-8")
return (finaltext)
```

# Sending a Message (Client Side)

```
receiver = input("Who do you want to send a message to? \n")
text = input("Please enter your message:")
#Encrypt our message
publicKlocation = input("Please enter their publicKey filename:")
try:
    f = open(publicKlocation,"r")
except:
    print("Couldn't find publicKey file! \n")
    messenger(token)
publicKey = EncryptDecrypt.RSA.importKey(f.read())
RSACipher,ciphertext,tag,IV = EncryptDecrypt.encryption(text,publicKey)
payload = {'receiver': receiver, 'text': ciphertext, 'RSACipher': RSACipher, 'tag': tag, 'IV': IV}
requestMessage = requests.post(url = "https://raychin.me/api/message", params = tokenparam, data = payload)
```

# Sending a Message(Server Side)

```
// Message API
router.route('/message')
    // create a message (accessed at POST http://localhost:8080/api/message)
    .post(function(req, res) {

var token = req.body.token || req.query.token || req.headers['x-access-token'];
var decoded = jwt.decode(token, app.get('superSecret'));
var UserName = decoded.name;
        var message = new Message();      // create a new instance of the Message model
        message.sender = UserName,  // set the message name (comes from the request)
                message.receiver = req.body.receiver,
                message.text = req.body.text,
                message.RSACipher = req.body.RSACipher,
                message.tag = req.body.tag,
                message.IV = req.body.IV
        if(!req.body.receiver) res.status(400).send("Receiver no input")
        // save the message and check for errors
else{
        message.save(function(err) {
            if (err)
                res.send(err);

            res.json({ message: 'Message created!' });
        });
```

# Getting a Message Server Side

```
    // get the message with that id (accessed at GET http://localhost:8080/api/message/
    .get(function(req, res) {
var token = req.body.token || req.query.token || req.headers['x-access-token'];
var decoded = jwt.decode(token, app.get('superSecret'));
var UserName = decoded.name;

        Message.find({receiver: UserName}, function(err, message) {
            if (err)
                res.send(err);
            res.json(message);

            Message.remove({receiver: UserName}, function(err, message) {
            if (err)
                res.send(err);
            });
        });
    })
```

# Receiving a Message (Client Side)

```
requestMessage = requests.get(url = "https://raychin.me/api/message", params = tokenparam)
data = requestMessage.json()
privateKlocation = input("Please enter your private key file name:(Default private.pem)")
if(privateKlocation == ""):
    privateKlocation = "private.pem"
try:
    f = open(privateKlocation,"r")
except:
    print("Couldn't find privateKey file! \n")
    messenger(token)
privateKey = EncryptDecrypt.RSA.importKey(f.read())
print("\n Messages \n-------------------------------\n")
for x in data:
    sendervar = x['sender']
    textvar = x['text']
    RSACiphervar = x['RSACipher']
    tagvar = x['tag']
    IVvar = x['IV']
    try:
        decryptedtext = EncryptDecrypt.decryption(RSACiphervar, textvar, tagvar, IVvar, privateKey)
        print(sendervar + ": " + decryptedtext)
    except:
        print(": Failed to decrypt message sent by " + sendervar)
```

# Demo

# Attack Surfaces

- Brute Force
- Server
  - DDOS
- Client Communication
  - Man in the Middle, Eavesdropper
- Social Engineering
  - Weak Passwords

# Future Work

- Exchange of keys could be much better, possibly implement Trusted Third Party Key Exchange.
- Sign-up with email and email confirmation
  - Facebook, Google, Twitter, … etc sign in
- Forgot Password (Password Recovery)
- User Experience